Self-stabilizing algorithms for the planarization problem in complete bipartite networks

Abstract

The maximum planarization problem asks to find a spanning planar subgraph having the largest number of edges. In this paper, we propose two simple self-stabilizing algorithms for this problem in complete bipartite graphs. The first one is an approximation algorithm; it finds a planar subgraph of approximation ratio 0.6 in the average case. Based on the techniques used in the first algorithm, the second algorithm finds a maximum planar subgraph. The time complexity is $O(n)$ rounds for both algorithms.

1 Introduction

A graph is *planar* if we can draw it on a plane without crossing edges. In this paper, we consider the *planarization* problem for *self-stabilizing systems*: Given a distributed system whose underlying topology is a connected and undirected graph G, we determine a subset of edges that induces a planar graph. The induced planar graph is said to be *maximal* if adding one more edge results in a non-planar graph. Among all maximal planar graphs, the one with the largest number of edges is said to be *maximum*. Finding a maximum planar subgraph is known to be an NP-complete problem [9], while finding a maximal one can be done in polynomial time [2, 4].

Planarization has many applications. For example, researchers have found that it can be applied to the routing problem if nodes know their geographic location [6, 7]. One such approach is *GPSR* (*Greedy Perimeter Stateless Routing*), which delivers packets in wireless networks [6]. The idea of GPSR is to repeatedly shorten the distance between a packet and its destination. When receiving a packet, a node forwards the packet to the neighbor closest to the packet destination by calculating *Euclidean distances*; thus the packet is forwarded in a greedy way. If such a neighbor does not exist, the node sends the packet by using the *right-hand rule*, which makes the packet travel along a *face* of the planar subgraph. As a result, the packet will be delivered to some node that is closer to the destination; that node then sends the packet via the greedy method again. By this way, the packet eventually arrives the destination.

The maximal planarization problem (i.e., finding a maximal planar subgraph) is highly relevant to the *planarity-testing* problem [5], which asks whether the input graph G is planar or not. Given a solution to one of these problems, we can use it to solve the other problem, as follows. Let V denote the node set of G.

Given a planarity-testing procedure, we can find a maximal planar subgraph G' by initializing $G' = (V, \emptyset)$ and then running a loop that repeatedly adds an edge e to G' if $G' \cup \{e\}$ is planar. When the loop terminates, G' is a maximal planar graph. On the other hand, given a procedure that outputs a maximal planar graph, we can use it to solve the planarity-testing problem by checking whether the input graph G equals to the output graph G'. If $G = G'$, then G is reported to be planar.

Instead of finding maximal planar graphs, people in the field of distributed computing tend to construct planar graphs that are dense enough. That is, the more planar edges, the better. In addition, the planar graphs should be constructed by *local algorithms* because each node has only a partial view of the system [10]. We also have to guarantee the connectivity of the constructed planar graph, in which there should not be disconnected components.

When every node knows its geometric location, there are many distributed ways to construct planar graphs from *unit disk graphs*. Such kinds of planar graphs include *relative neighborhood graph* (RNG) [6], *Gabriel graph* (GG) [6], and *k-localized Delaunay triangulation* ($\text{LDel}^{(k)}$), where $k \geq 1$ [8]. The graphs RNG and GG are constructed by the following methods. Let $dist(u, v)$ be the *Euclidean distance* between two nodes u and v. An edge (u, v) is called RNG edge if every other node w satisfies $dist(u, v) \leq dist(w, u)$ or $dist(u, v) \leq dist(w, v)$. The RNG edges induce RNG. Similarly, an edge (u, v) is said to be a *Gabriel edge* if no other node locates inside the circle of the diameter $\overline{uv}$. The Gabriel edges induce GG. By definition, the relative neighborhood graph is a subgraph of Gabriel graph since any RNG edge is also a Gabriel edge.

The definition of $LDel^{(k)}$ is as follows. A triangle $\triangle uvw$ is said to be a *k-localized Delaunay triangle* if its circumcircle contains no any other node x less than or equal to k hops away from u, or v, or w. That is, any node x, if any, inside the circumcircle of $\triangle uvw$ is more than k hops away from the three nodes. The k-localized Delaunay graph is induced by Gabriel edges and k-localized Delaunay triangles. For $2 \leq k \leq n$, $LDel^{(k)}$ is a planar, but $LDel^{(1)}$ is not necessarily planar, where n is the number of nodes. The relation between these planar graphs is $\text{RNG} \subseteq \text{GG} \subseteq LDel^{(n)} \subseteq LDel^{(n-1)} \subseteq \ldots \subseteq LDel^{(1)}$ [8]. The literature [8] gives a geometric illustration showing how to find all these graphs.

In this paper, we propose two algorithms for the planarization problem. We focus on complete bipartite graphs without location information of the nodes. The first algorithm is an approximated one; it finds a planar graph having at least $(3n - \sqrt{n^2 - 4m})/2 - 2$ edges, where n is the number of nodes and m is the number of edges in G. The second algorithm finds a maximum planar graph. The idea behind both algorithms is to carefully identify a set of edges that induces a graph without a *subdivision* of K_5 or $K_{3,3}$, where K_5 is the complete graph of 5 nodes and $K_{3,3}$ is the complete bipartite graph with each partite set of 3 nodes. According to *Kuratowski's Theorem*, the induced graph is planar [11].

Our algorithms have the property of self-stabilization [3], which is a powerful fault-tolerant paradigm for distributed systems. Even when the system is in an illegal configuration, our algorithms can automatically bring it into a legal configuration within $O(n)$ rounds.

The rest of this paper is organized as follows. Section 2 introduces the system model and the terminologies. Sections 3 and 4 present the approximation algorithm and the maximum planarization algorithm, respectively. Finally, section 5

concludes this paper.

2 Preliminaries

In this section, we first introduce the model of self-stabilizing distributed systems. We then recall some terminologies and notation concerning graph theory.

2.1 System Model

A distributed system is modeled by a connected, undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. In this paper, we consider a complete bipartite network; that is, G is a complete bipartite graph K_{n_1,n_2}, where n_1 and n_2 are the sizes of the partite sets. We also consider an ID-based network; every node is hard-wired with a unique ID that never changes. Two nodes u and v are said to be neighbors if $(u, v) \in E$. Each node keeps a set of variables to represent its local state. A node can read and write to the variables of its own but can only read the variables of its neighbors.

We use the term *configuration* to represent a system state. Given a configuration c and its successor c', the transition from c to c' is called a *computation step*, denoted by $c \to c'$. The *computation* of the system is expressed by a sequence of configurations $c_0 \to c_1 \to \ldots$, where c_0 is an arbitrary initial configuration. We use $c_k \rightsquigarrow c_{k+k'}$ to denote a sequence of k' consecutive computation steps, where $k \geq 0$ and $k' \geq 0$.

Self-stabilization is a powerful fault-tolerant mechanism for distributed systems [3]. When local states of nodes change unexpectedly (by memory corruptions, for example,) the system configuration may become incorrect and we say that the system undergoes *transient faults*. A self-stabilizing system copes with transient faults by the following reasoning. All configurations are classified into two categories: *legitimate* and *illegitimate*. Starting from any initial configuration, a self-stabilizing system guarantees to be able to converge to and then remain in legal configurations. Once unexpected transient faults occur, it is considered to be in an initial configuration. By the property of self-stabilization, it will return to legal configurations. Formally speaking, let $\mathbb{S}$ denote all possible configurations and $\mathbb{S}_L$ be the set of legitimate configurations. For every $c_0 \in \mathbb{S}$, any computation of the system is $c_0 \rightsquigarrow c_\ell \rightsquigarrow c_{\ell+k}$, where $c_{\ell+k} \in \mathbb{S}_L$ for a finite integer $\ell \geq 0$ and any $k \geq 0$. The process $c_0 \rightsquigarrow c_\ell$ is called *convergence*, whereas $c_\ell \rightsquigarrow c_{ell+k}$ is called *closure*. The stabilization time is the time span from c_0 to the first legal configuration c_ℓ.

We encode the proposed algorithms by a set of rules. The format of a rule is "*guard* $\to$ *action*", where *guard* is a boolean formula and *action* is a set of program statements. When the guard of a rule is satisfied for a node, the node is said to be *privileged* and the rule is said to be *enabled*. The privileged node could execute the enabled rule by performing the corresponding action. A node executes a rule in an atomic way: it uninterruptedly evaluates the guard and performs the corresponding action.

There is a *distributed daemon* that decides the computation of the system. At the beginning of each computation step $c \to c'$, the daemon chooses a non-empty set of privileged nodes in c. Every of the chosen nodes executes a rule. When

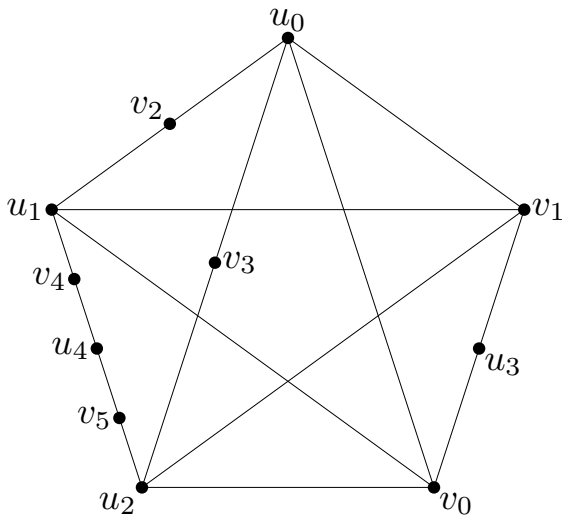

Figure 1: A subdivision of K_5.

the nodes finish their actions, the system is in the configuration c' and the next computation step begins.

2.2 Graph Theory Background

We assume that the reader is familiar with fundamental knowledge about graph theory. Here we recall some terminologies and notations that are used less frequently.

Inducing Let $G = (V, E)$ be a graph and E' be a subset of E. The graph induced by E' is a subgraph of G, denoted by $G[E']$, that is composed of E' and the nodes connected by E'. Similarly, let V' be a subset of V. The graph induced by V' is composed of V' and the edges whose endpoints $\in V'$.

Subdivision A subdivision of an edge (u, v) is an operation that replaces the edge with a path u, w, v, where w is a new node. A subdivision of a graph G is a graph obtained by repeatedly applying edge subdivisions to G. Figure 1 shows an example of a subdivision of K_5. That figure also demonstrates a property that a bipartite graph may have a subdivision of K_5 (note that $\{u_i | 0 \leq i \leq 4\}$ and $\{v_i | 0 \leq i \leq 5\}$ are partite sets.)

There is a known characteristic concerning planar graphs: Kuratowski's theorem. This theorem states that G is planar if and only if G does not contain a subdivision of K_5 or $K_{3,3}$. Based on this theorem, the idea of searching for planar subgraphs becomes a little more straightforward. For not inducing a subdivision of K_5, there should not be 10 internally disjoint paths completely interconnecting 5 nodes. Similarly, for not inducing a subdivision of $K_{3,3}$, there should not be 9 internally disjoint paths completely connecting 3 nodes with another 3 nodes.

Theorem 1 (Kuratowski)**.** *A graph is planar if and only if it does not contain a subdivision of K_5 or $K_{3,3}$.*

3 The Approximation Algorithm

In this section, we present an approximation planarization algorithm. We assume a complete bipartite network $K_{n_1,n_2} = (V, E)$, where n_1 and n_2 are the sizes of the partite sets. We use G and K_{n_1,n_2} interchangeably and use the notation $v < u$ to express that the ID of v is lower than that of u. In addition, we use $n = n_1 + n_2$ and $m = n_1 n_2$ to denote the number of nodes and edges, respectively.

In subsection 3.1, we explain the basic idea and show how to find a subset $\hat{E}$ of edges that induces a planar graph. In subsection 3.2, we prove the planarity of $G[\hat{E}]$, the time complexity, and the performance ratio of the algorithm.

3.1 The algorithm

For constructing a planar graph, we carefully select a set $\hat{E}$ of edges so that $G[\hat{E}]$ has no subdivision of K_5 or $K_{3,3}$. The definition of $\hat{E}$ is as follows. Each node u keeps a variable S, denoted by $S.u$, to maintain a list of neighbors as its *successors*. The set $\hat{E}$ is then defined to be

$$\hat{E} = \{(u, v) | u \in V, v \in S.u\}.$$

In the rest of this paper, if $v \in S.u$ holds, v is said to be a successor of u and u is said to be a *predecessor* of v.

The overview of the proposed algorithm is as follows. We first give a label to every node. Ideally, the value of a label is 0, 1, or 2, so the node set V is partitioned into non-empty, disjoint sets V_0, V_1 and V_2. The union of V_0 and V_2 is one partite set of G, whereas V_1 is the other partite set. Moreover, we makes $|V_0| = 1$. Depending on the set it belongs to, a node has a corresponding way to decide its successors. The node in V_0 sets $S = \emptyset$, and the nodes in V_1 set S to V_0. The other nodes group their neighbors to form a family of *candidate sets*, each of which consists of two neighbors. They decide their successors by simply selecting one of those candidate sets.

To assign the labels to the nodes, we first run the BFS spanning tree algorithm in [1]. Thus each node maintains the necessary variables needed for [1]. Among them, there is a variable d that records the depth of a node in the tree. When the spanning tree algorithm stabilizes, there are only three possible values for the variable d: 0, 1, and 2. The root node, which is the node of maximum ID, has d=0; the neighbors of the root have $d = 1$; the others have $d = 2$. The value of d represents the node label, and we define that a node u belongs to V_i if $d.u = i$, where $i \in \{0, 1, 2\}$.

Nodes decide their successors based on labels. The general principle is to let nodes of bigger labels select those of smaller labels as successors. The node in V_0 thus has no successor, whereas every node in V_1 has only one successor, for $|V_0|$ =1. Those nodes in $V_0 \cup V_1$ induces a subgraph $K_{1,|V_1|}$ so we have to deal with nodes in V_2 in particular in order not to produce subdivisions of $K_{3,3}$ and subdivisions of K_5.

For this, we demand that every node in V_2 has only two successors. We organize V_1 into $|V_1| - 1$ *candidate sets*. Every candidate set has exactly two nodes (except for the case $|V_1| = 1$), according to node IDs in the ascending order. Every node in V_2 simply selects a candidate set to be its successor set.

Definition 1. *Let u be any node and let v_0, v_1,...,v_{k-1} denote u's neighbors sorted in the ascending order. The family of candidate sets for node u, denoted by $FCS.u$, is as follows:*

$$FCS.u \equiv \begin{cases} \{\{v_i | 0 \leq i < k, d.v_i = 0\}\} & \text{if } d.u = 1 \\ \{\{v_0\}\} & \text{if } d.u = 2 \wedge k = 1 \\ \{\{v_i, v_{i+1}\} | 0 \leq i \leq k-2\} & \text{if } d.u = 2 \wedge k > 1 \\ \{\emptyset\} & \text{otherwise} \end{cases}$$

Now we begin to present our algorithm. It has two layers in a sense. The lower layer is to determine a correct label for each node by running the spanning tree algorithm in [1]. The upper layer is to choose planar edges; it has only one rule R0. Every node always sets S to one of its candidate sets.

R0: $S.u \notin FCS.u$
$\rightarrow$ Set $S.u$ to some candidate set $\in FCS.u$;

The induced graph $G[\hat{E}]$ has $|V_1| + 2|V_2| - 2$ edges. It is because every node in V_1 corresponds to 1 edge in $G[\hat{E}]$ and every node in V_2 corresponds to 2 edges. In the next subsection, we will show that $G[\hat{E}]$ is planar and $|\hat{E}| \geq (3n - \sqrt{n^2 - 4m})/2 - 2$.

3.2 Correctness and Time Complexity

In this subsection, we prove the convergence property and the time complexity for the approximation algorithm. From the analyses of [1], the variable d stabilizes in $O(K + (deg \times dia)) = O(n + ((n-1) \times 2)) = O(n)$ rounds, where K is the maximum possible number of nodes, deg is the maximum degree and dia is the diameter of G (When we say that the variable d stabilizes, we actually mean that the value of d maintained by every node will no longer change, until transient faults occur.) What remains is to show that (1) eventually there is no privileged node, and then, (2) $G[\hat{E}]$ is planar.

Lemma 1. *Once the variable d has stabilized, each node executes R0 at most once.*

Proof. Because the variable d has stabilized, each node has a fixed family of candidate sets. Therefore, after a node u executes R0, $S.u \in FCS.u$ holds and u no longer executes R0 henceforth. □

Because it takes $O(n)$ rounds for the variable d to stabilize and another one round for the variable S to stabilize, we have the following lemma.

Lemma 2. *Starting from any configuration, there will be no privileged node after $O(n)$ rounds .*

Lemma 2 states that the system will converge to and then remain in some configuration. In the next lemma, we show that $G[\hat{E}]$ is planar in that configuration.

Lemma 3. *When no node is privileged, $G[\hat{E}]$ is a spanning planar graph.*

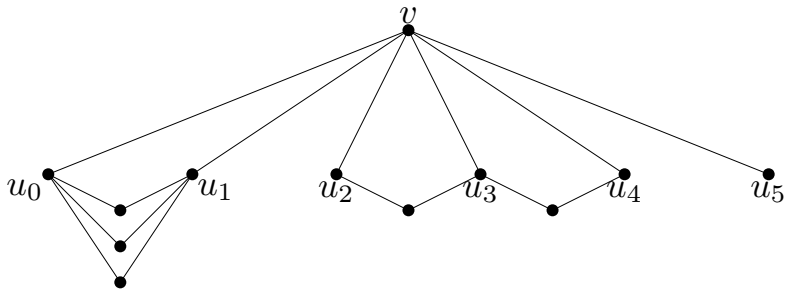

Figure 2: A drawing of $G[\hat{E}]$.

Proof. We prove this lemma by giving a drawing of $G[\hat{E}]$.

We first put the node $v \in V_0$ arbitrarily on the plane; below v we put nodes in V_1 horizontally, from left to right according to their IDs in the ascending order. Finally, for any node in V_2 we put the node between its successors. With such arrangement, we can draw $G[\hat{E}]$ without crossing edges. Figure 2 gives an example of a drawing of $G[\hat{E}]$. In the figure, $V_0 = \{v\}$, $V_1 = \{u_i | 0 \leq i \leq 5\}$, and V_2 is the set of untagged nodes. □

Now, we begin to analyze the number of edges in $|\hat{E}|$. Without loss of generality, we assume that the node in V_0 belongs to the partite set of size n_2. This assumption implies $|V_1| = n_1$, $|V_2| = n_2 - 1$, and $|\hat{E}| = n_1 + 2(n_2 - 1) = n_1 + 2n_2 - 2$. These equations suffice to find $|\hat{E}|$:

Lemma 4. $|\hat{E}| \geq \frac{3n-\sqrt{n^2-4m}}{2} - 2$

Proof. From the equations $n_1 + n_2 = n$ and $n_1 n_2 = m$, we can find

$$n_2 = \frac{n \pm \sqrt{n^2 - 4m}}{2}$$

and thus

$$\begin{aligned} |\hat{E}| &= n_1 + 2n_2 - 2 \\ &= n + n_2 - 2 \\ &\geq \frac{3n-\sqrt{n^2-4m}}{2} - 2 \end{aligned}$$

□

Theorem 2. *The approximate algorithm finds a spanning planar graph with* $(3n - \sqrt{n^2 - 4m})/2 - 2$ *edges in* $O(n)$ *rounds.*

Proof. It is a direct consequence of lemma 2, lemma 3 and lemma 4. □

The performance ratio is different for the worst case and the average case. When n_2 is a constant, $|\hat{E}|$ is just a few more than $n-1$, the number of edges in the simplest spanning planar graph: spanning tree. In other words, the performance ratio is about $1/2$ in the worst case, since G is triangle-free and its maximum planar graph has at most $2n - 4$ edges. On the other hand, the performance ratio is about 0.6 in the average case. The key idea is to find the average value for m. Since $1 \leq n_1 \leq n - 1$, m has $n - 1$ possible values so its average is

$$\sum_{\substack{n_1+n_2=n; \\ n_1,n_2 \geq 1}} \frac{n_1 n_2}{n-1} = \sum_{k=1}^{n-1} \frac{k(n-k)}{n-1} = \frac{n^2+n}{6}$$

From the above equation, we can infer that the average of $|\hat{E}|$ is

$$\begin{aligned} & (3n - \sqrt{n^2 - 4(n^2+n)/6})/2 - 2 \\ \geq\ & (3n - n/\sqrt{3})/2 - 2 \\ =\ & \tfrac{9-\sqrt{3}}{6}n - 2 \end{aligned}$$

Theorem 3. *The performance ratio of the approximate algorithm is* $(9+\sqrt{3})/12 \approx 0.6$ *in the average case.*

Proof. Since the maximum planar subgraph has at most $2n-4$ edges and since the average number of edges in $G[\hat{E}]$ is $(9-\sqrt{3})n/6 - 2$, we have

$$\text{performance ratio} \geq \frac{(9-\sqrt{3})n/6 - 2}{2n-4} \geq \frac{9-\sqrt{3}}{12} \approx 0.6$$

in the average case. □

We can further improve the algorithm and find a truly maximum spanning planar graph. In the next section we show how to do this.

4 The Maximum Planarization Algorithm

4.1 The Algorithm

In this section, we show how to construct a maximum spanning graph from a complete bipartite graph K_{n_1,n_2}. Here we only consider the case that $n_1, n_2 \geq 2$; otherwise G is planar. Similar to the algorithm proposed in section 3, we first run the BFS spanning tree algorithm in [1] and every node maintains a variable d. When the tree construction algorithm stabilizes, the value of d is 0, 1, or 2.

Every node keeps a variable S for representing their successors and defining planar edges, similar to what we did before. Nodes in V_0 have no successors, while nodes in V_1 (resp., V_2) select nodes in V_0 (resp., V_1) to be their successors.

Unlike the approximation algorithm in section 3, we organize nodes into V_0, V_1 and V_2 in a different way. In addition to the node of $d = 0$, the set V_0 also contains the node that has the maximum ID among nodes of $d = 2$. By this setting, V_0 contains two nodes, V_1 contains nodes of $d = 1$, and V_2 contains the other nodes.

Since we move a node of $d = 2$ from V_2 into V_0, we have to let the node know it is in V_0 rather than in V_2. This could be done through nodes in V_1. All nodes in V_1 select a neighbor of $d = 0$ and another neighbor of $d = 2$ with the maximum ID to be their successors. If a node of $d = 2$ finds that all the neighbors are its predecessors, then it is aware that it belongs to V_0 and should reset its S to $\emptyset$. Following this logic, the definition of candidate sets changes accordingly.

Definition 2. *Let u be any node and let v_0, v_1,...,v_{k-1} denote u's neighbors sorted in the ascending order. The family of candidate sets for node u, denoted by $FCS.u$, is*

$$FCS.u \equiv \begin{cases} \{\{v_i | 0 \leq i < k, d.v_i = 0\} \cup \{v_{k-1}\}\} & \text{if } d.u = 1 \\ \{\{v_i, v_{i+1}\} | 0 \leq i \leq k-2\} & \text{if } (d.u = 2) \\ & \quad \wedge (\forall v_i : 0 \leq i < k \Rightarrow u \notin S.v_i) \\ \{\emptyset\} & \text{otherwise} \end{cases}$$

The self-stabilizing maximum planarization algorithm is as follows. It is two layered in a sense. The lower layer is for determining the variable d by the spanning tree algorithm in [1]. The upper layer is to choose planar edges by rule R0:

R0: $S.u \notin FCS.u$
$\rightarrow$ Set $S.u$ to some candidate set $\in FCS.u$;

According to our design, only two nodes have empty candidate set so they have no successors. The other nodes have exactly two successors. The induced graph thus has $2(n-2) = 2n-4$ edges.

4.2 Correctness and Time Complexity

In this subsection, we show that the algorithm stabilizes and does find a maximum planar graph. The proofs are similar to those in subsection 3.2. We show that the algorithm converges to a configuration in which there is no privileged node (lemmas 5 and 6). We then show that the induced graph is planar and maximum in that configuration (lemma 7).

Lemma 5. *Once the variable d has stabilized, each node executes R0 at most twice.*

Proof. Let u denote the node that has the maximum ID among nodes of $d = 2$. We show that u executes R0 at most twice and the other nodes execute R0 at most once.

By the assumption that the variable d has stabilized, every node except u has a fixed family of candidate sets, so it executes R0 at most once. For node u, $FCS.u$ changes after all its neighbors appoint u as their successor. After $FCS.u$ changes, node u gains one more privilege to execute R0 so it executes R0 at most twice. □

Lemma 6. *Starting from any configuration, there will be no privileged node after $O(n)$ rounds.*

Proof. According to [1], the variable d stabilizes in $O(n)$ rounds. By lemma 5, no node is privileged in another two rounds. Thus this lemma holds. □

Lemma 7. *When there is no privileged node, $G[\hat{E}]$ is a maximum spanning planar graph.*

Proof. We show that $G[\hat{E}]$ is planar and has $2n-4$ edges. That suffices to prove this lemma since a triangle-free planar graph has at most $2n-4$ edges.

We give a drawing to show that $G[\hat{E}]$ is planar. First, put nodes in V_1 horizontally, from left to right, according to their IDs in the ascending order. Then, put nodes in V_2 between their successors. Finally, the two nodes in V_0 are put at the top and the bottom of the plane, respectively. After deciding the positions of nodes, we can draw $G[\hat{E}]$ without crossing edges. Figure 3 illustrates an example of a drawing.

From the property that every node except for the two in V_0 has two successors, there are $2(n-|V_0|) = 2(n-2) = 2n-4$ edges in $G[\hat{E}]$. Since G is triangle-free, its maximum planar subgraph has at most $2n-4$ edges. Therefore, $G[\hat{E}]$ is a maximum planar subgraph. □

Theorem 4. *The algorithm stabilizes in $O(n)$ rounds.*

Proof. It is a direct consequence of lemma 6 and lemma 7. □

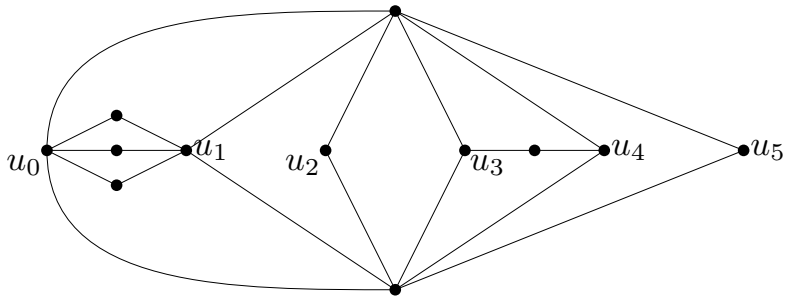

Figure 3: An example of a maximum planar graph; $V_1 = \{u_i | 0 \leq i \leq 5\}$.

5 Conclusion

In this paper, we propose two self-stabilizing algorithms for the planarization problem; our approach heavily depends on the property of complete bipartite graphs. By defining candidate sets, the first algorithm constructs a planar subgraph with approximation ratio 0.6 in the average case, while the second algorithm finds a maximum planar subgraph. Both of them have linear time complexity; they construct planar graphs within $O(n)$ rounds.

In distributed systems, it is not easy to find a spanning planar graph with non-trivial approximation ratio. One reason is that it is not easy to preserve connectivity because each node uses only local information to decide planar edges. Another reason is that it is very likely to form subdivisions of K_5 and $K_{3,3}$. The latter reason could be seen through the following example. Assume that we already have a simplest spanning planar graph (i.e., spanning tree). For any 6 nodes in that induced graph, we can construct a subdivision of $K_{3,3}$ by adding just 9 more edges, as follows. First we partition the 6 nodes into two sets, each of which has 3 nodes. For each pair (u, v) of nodes in different sets, we connect a descendant of u and another descendant of v. As a result, there is one more path connecting u and v. The other 8 edges are created in a similar way. If all those 9 paths are internally disjoint, then a subdivision of $K_{3,3}$ is constructed. (Actually, it is possible to form a subdivision of $K_{3,3}$ by adding just 6 edges to a spanning tree.)

The research on the planarization problem is still active. It is interesting to find solutions with non-trivial approximation ratio (i.e., 1/2 for triangle-free graphs and 1/3 for other cases.) Moreover, it is also interesting to consider graphs that are less regular.

References

[1] A. Arora and M. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, 1994.

[2] L. Cai, X. Han, and R. E. Tarjan. An $o(m \log n)$-time algorithm for the maximal planar subgraph problem. *SIAM Journal on Computing*, 22:1142–1162, 1993.

[3] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[4] H. N. Djidjev. A linear-time algorithm for finding a maximal planar subgraph. *SIAM journal on discrete mathematics*, 20(2):444–462, 2006.

[5] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

[6] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, 2000.

[7] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[8] X. Y. Li, G. Calinescu, P. J. Wan, and Y. Wang. Localized delaunay triangulation with application in ad hoc wireless networks. *IEEE transactions on parallel and distributed systems*, 14(10):1035–1047, 2003.

[9] A. Liebers. Planarizing graphs — a survey and annotated bibliography. *Journal of Graph Algorithms and Applications*, 5(1):1–74, 2001.

[10] J. Urrutia. Local solutions for global problems in wireless networks. *Journal of Discrete Algorithms*, 5(3):395–407, 2007.

[11] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2001.